2021.07.10. 교육

나정휘

https://justiceHui.github.io/

목차

- Binary Search Tree
- Heap
- Union-Find (Disjoint Set)
- Small to Large
- Sparse Table
- Lowest Common Ancestor

이진 탐색 트리

이진 탐색

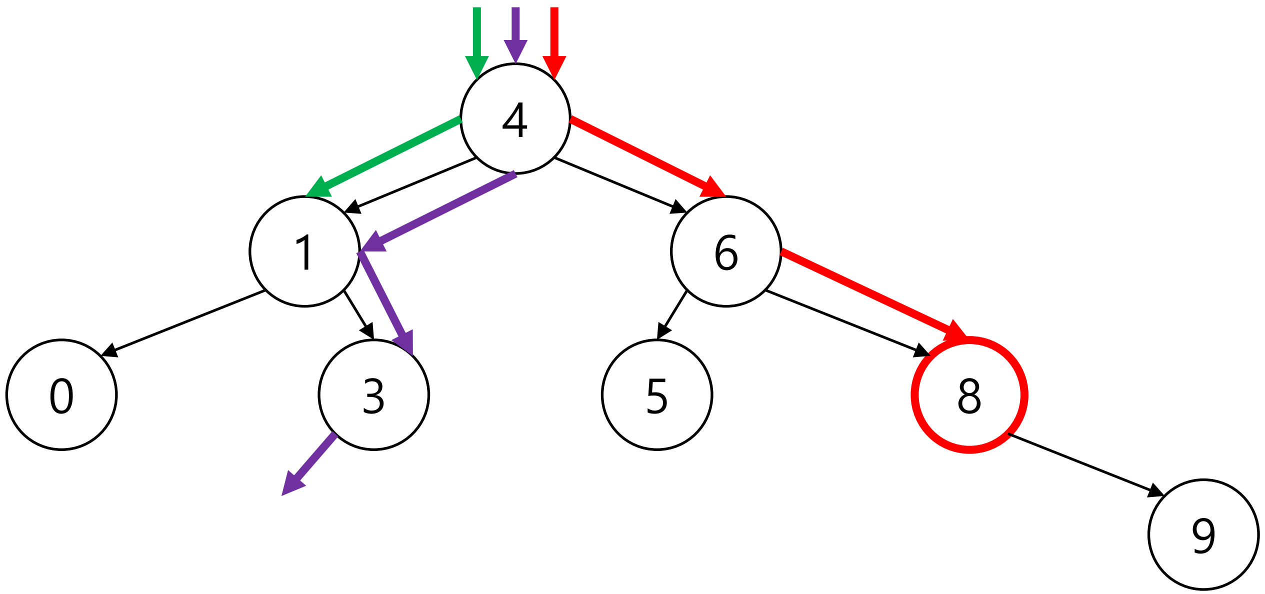
- 구간 [l, r]에서 어떤 값 x가 있는지 탐색
  - 중간 지점 m = (l + r) / 2 잡고
  - A[m] = x이면 탐색 완료
  - A[m] < x이면 x는 m보다 오른쪽에 있음 : l = m + 1
  - A[m] > x이면 x는 m보다 왼쪽에 있음 : r = m – 1

이진 탐색

- A = {0 1 3 4 5 6 8 9}, x = 8
  - l = 0, r = 7, m = 3, A[m] = 4
  - l = 4, r = 7, m = 5, A[m] = 6
  - l = 6, r = 7, m = 6, A[m] = 8
  - 종료
- x = 1
  - l = 0, r = 7, m = 3, A[m] = 4
  - l = 0, r = 2, m = 1, A[m] = 1
  - 종료

이진 탐색 트리

- 이진 탐색 과정을 이진 트리로 나타낸 것
- x = 8
- x = 1
- x = 2

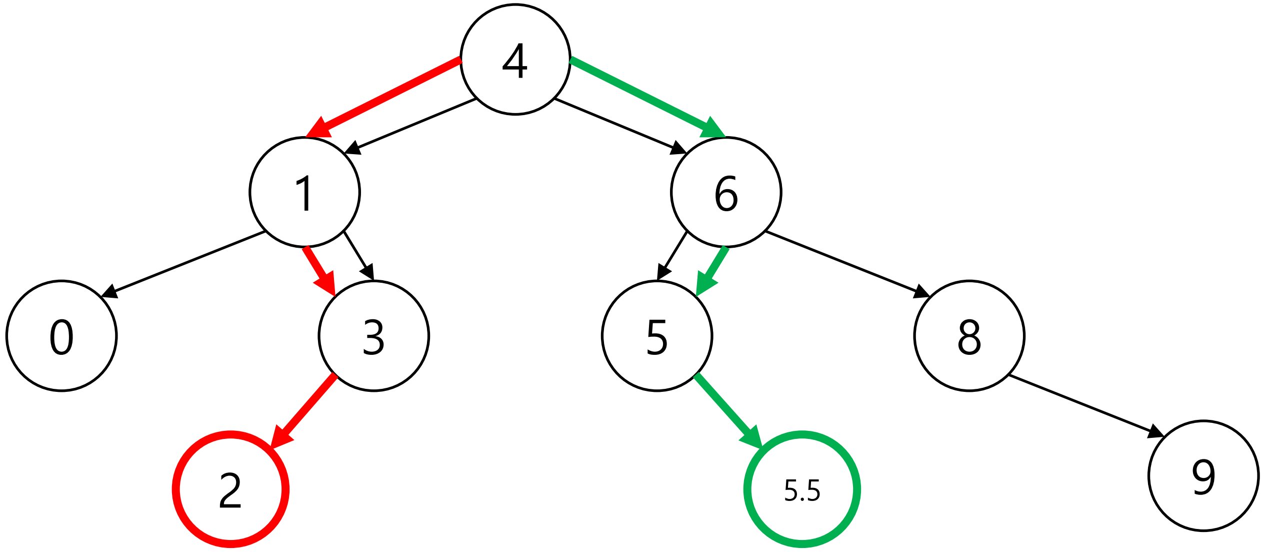


질문?

원소 삽입

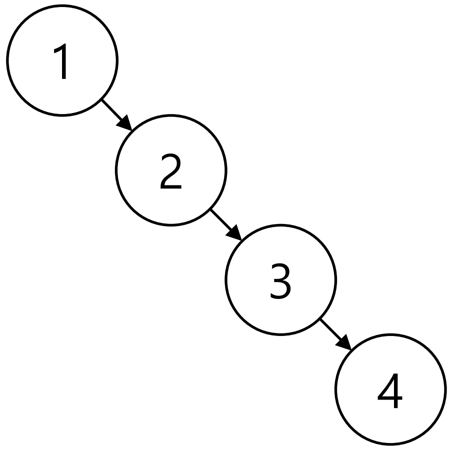
- 루트부터 시작
  - 현재 정점의 원소보다 크면 오른쪽 / 작으면 왼쪽
  - 리프에 도달할 때까지 반복
  - 리프의 자식으로 추가

- INSERT 2
- INSERT 5.5



시간 복잡도

- 트리의 높이를 h라고 하면 O(h)
- 최악의 경우 O(N)
  - INSERT 1
  - INSERT 2
  - INSERT 3
  - INSERT 4
  - …



- 최선의 경우 O(log N)
  - Why?

원소 삭제

- 귀찮음
- 굳이 알 필요 없음 (아마도?)

- 원소 삭제도 O(h)

가장 큰/작은 원소

- 가장 큰 원소
  - 루트에서 시작해서
  - 오른쪽 자식으로 계속 내려감

- 가장 작은 원소
  - 루트에서 시작해서
  - 왼쪽 자식으로 계속 내려감

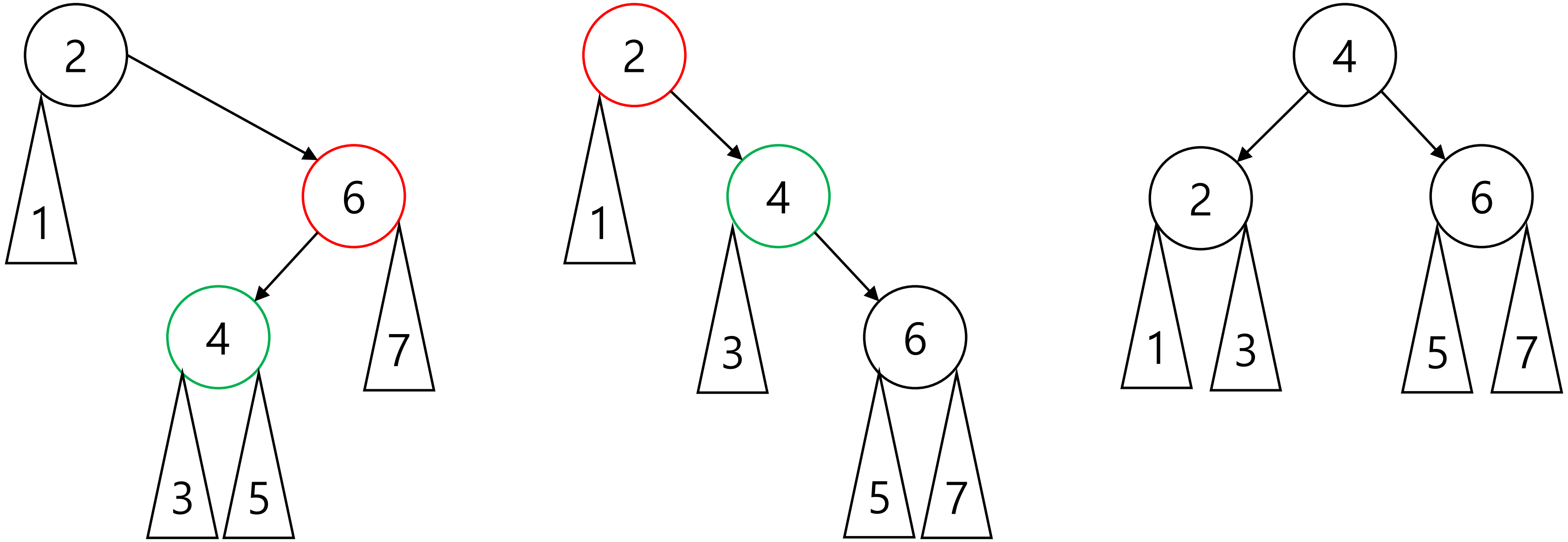
중위 순회(Inorder Traversal)

- 이진 탐색 트리의 중위 순회는 원소를 오름차순으로 순회함
  - 왼쪽 서브 트리(자신보다 작은 원소)를 모두 순회한 뒤
  - 루트(자신)을 보고
  - 오른쪽 서브 트리(자신보다 큰 원소)를 순회함

질문?

시간 복잡도 개선

- 단순하게 구현하면 최악의 경우 O(N)을 피할 수 없음
- Balanced Binary Search Tree(BBST)
  - 이진 탐색 트리를 유지하면서 높이를 O(log N)으로 유지할 수 있음
  - 대충 이런 식으로...
  - 굳이 알 필요 없음 (아마도?)



BBST의 종류

- 알아두면 언젠가 쓸모 있음 ex. 면접
  - AVL Tree
  - B-Tree
  - Red Black Tree
  - Treap
  - Splay Tree
  - …

std::set

- set : 집합
- 집합을 관리하는 STL
  - 원소의 중복을 허용하지 않음 (std::multiset은 중복 허용)
  - 원소를 오름차순으로 관리
  - 원소 삽입/삭제/검색 가능
  - 오름차순/내림차순 순회 가능

  - 보통 Red Black Tree로 구현되어 있음
    - 원소 삽입/삭제/검색 O(log N)에 동작

std::map

- key-value pair를 관리하는 STL
  - key의 중복을 허용하지 않음
  - key의 오름차순으로 관리
  - key 삽입/삭제/검색 가능
  - value 변경 가능
  - key 오름차순/내림차순 순회 가능 (value도 함께)

  - 보통 Red Black Tree로 구현되어 있음
    - O(log N)

BOJ 1822 차집합

- std::set

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int N, M; cin >> N >> M;
    set<int> A;
    for(int i=0; i<N; i++){
        int x; cin >> x;
        A.insert(x);
    }
    for(int i=0; i<M; i++){
        int x; cin >> x;
        if(A.find(x) != A.end()) A.erase(x);
    }
    cout << A.size() << "\n";
    for(auto i : A) cout << i << " ";
}
```

BOJ 20920 영단어 암기는 괴로워

- std::map

```
#include <bits/stdc++.h>
using namespace std;

int N, M;
vector<string> V;
map<string, int> C;

bool Compare(const string &a, const string &b){
    if(C[a] != C[b]) return C[a] > C[b];
    if(a.size() != b.size()) return a.size() > b.size();
    return a < b;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M;
    for(int i=0; i<N; i++){
        string s; cin >> s;
        if(s.size() >= M) V.push_back(s), C[s]++;
    }
    sort(V.begin(), V.end(), Compare);
    for(int i=0; i<V.size(); i++){
        if(i == 0 || V[i-1] != V[i]) cout << V[i] << "\n";
    }
}
```

질문?

힙

가장 큰 원소를 찾는 문제

- 아래 3가지 작업을 수행할 수 있는 자료구조를 생각해보자.
  - 원소 삽입
  - 가장 큰 원소 탐색
  - 가장 큰 원소 제거

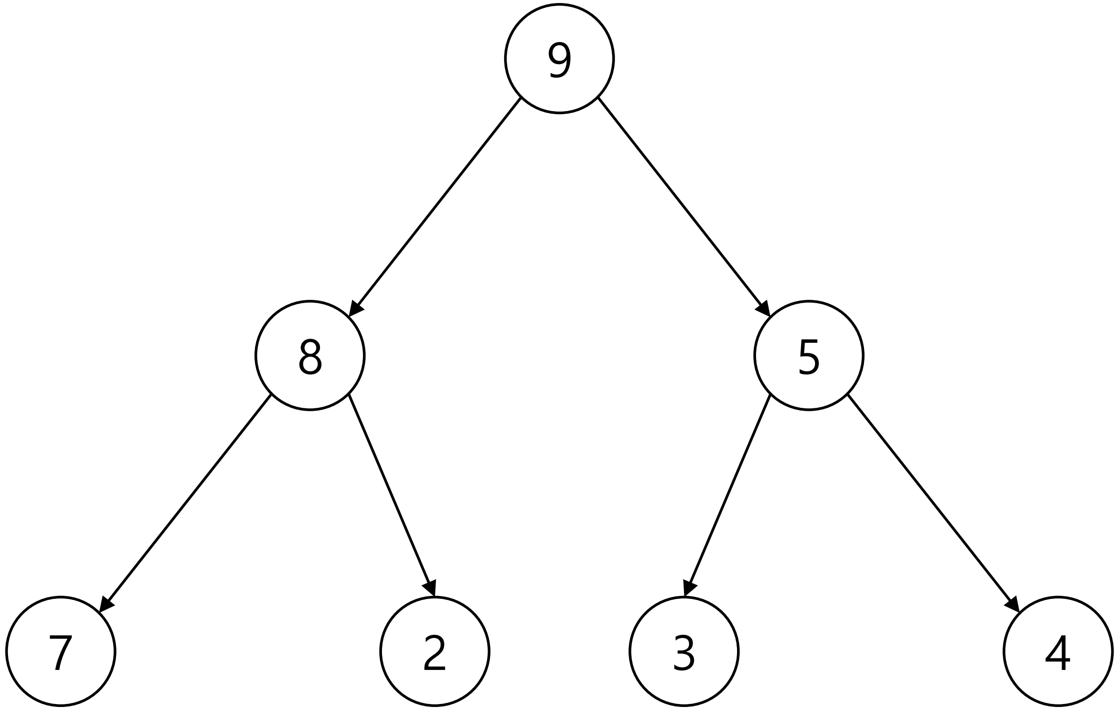
- BBST를 만들고 가장 오른쪽 자손을 본다?
  - BBST는 O(log N)이지만 상당히 느림
  - log N와 10 * log N 모두 O(log N)이지만 10 * log N이 훨씬 느림

힙

- 각 정점의 값은 자식 정점들의 값보다 크다.
  - 이진 탐색 트리보다 제약 조건이 약함
  - 보통 제약 조건이 약하면 더 효율적으로 구현 가능

- 완전 이진 트리(Complete Binary Tree) 형태이다.
  - 배열로 구현 가능
  - 루트 : 1번 인덱스
  - v의 부모 : v / 2
  - v의 왼쪽/오른쪽 자식 : v*2, v*2+1

힙



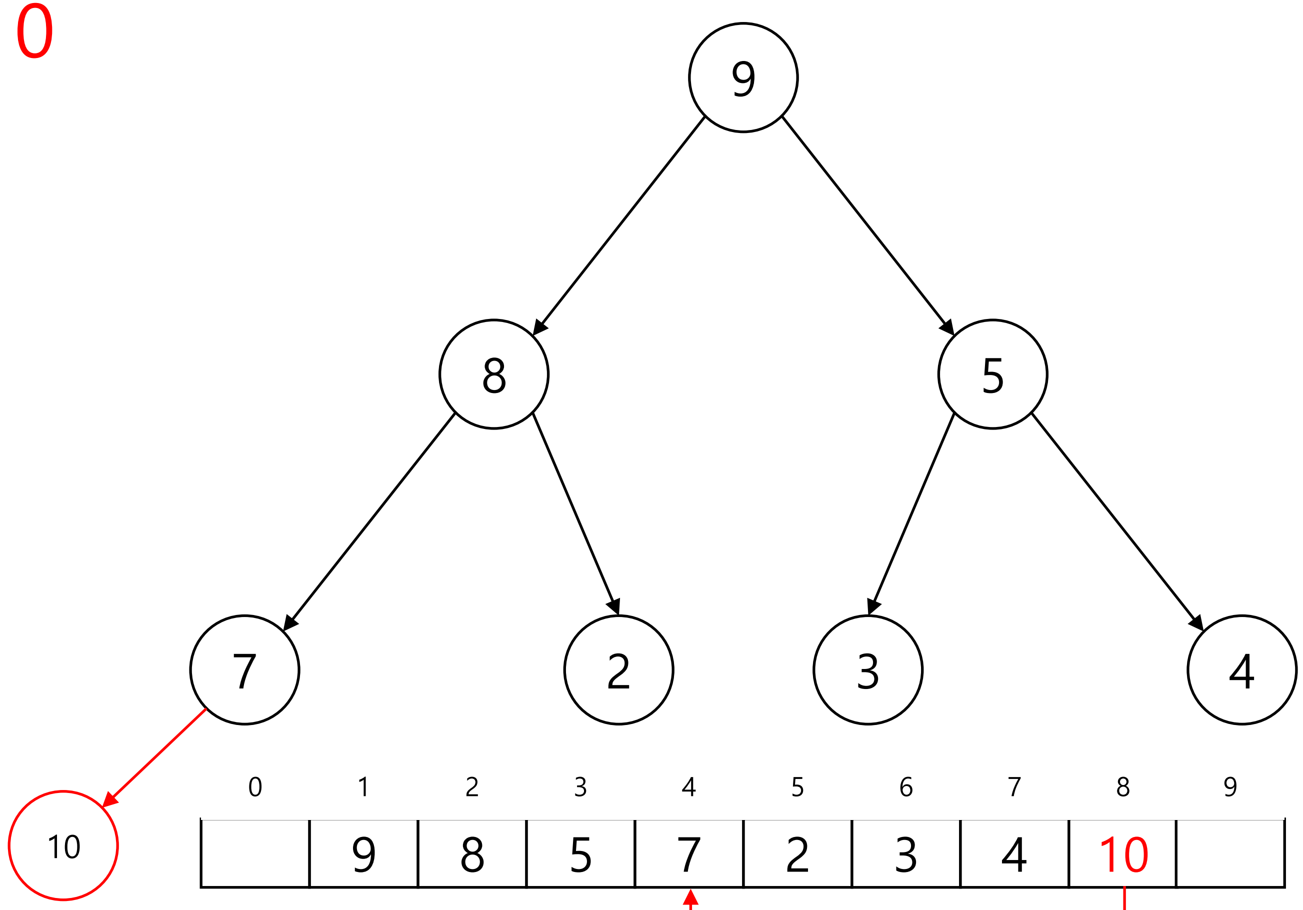
0	1	2	3	4	5	6	7	8	9
	9	8	5	7	2	3	4		

원소 삽입

- 맨 뒤에 원소 추가
- 만약 부모보다 작으면 부모와 교환

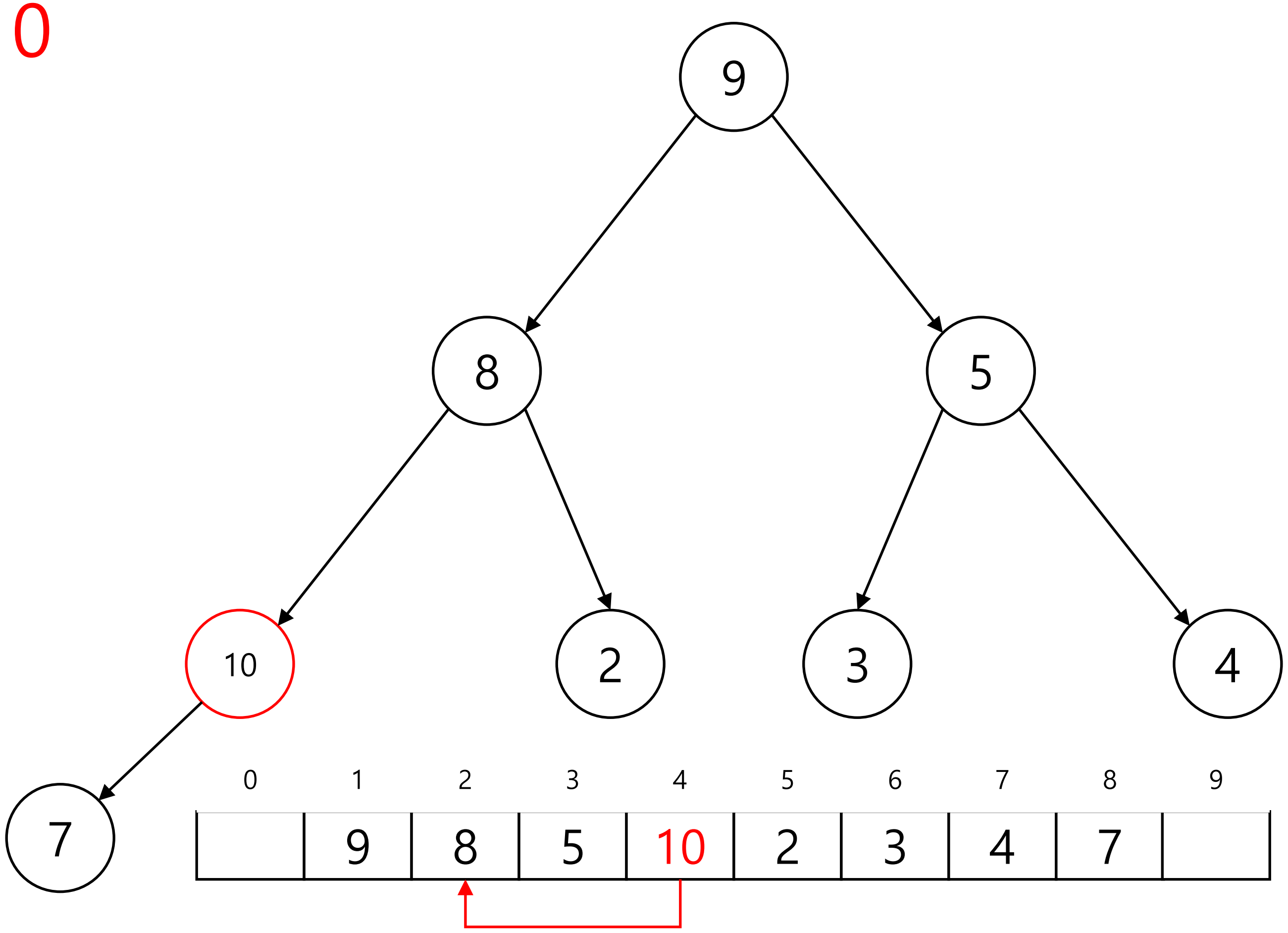
원소 삽입

- INSERT 10



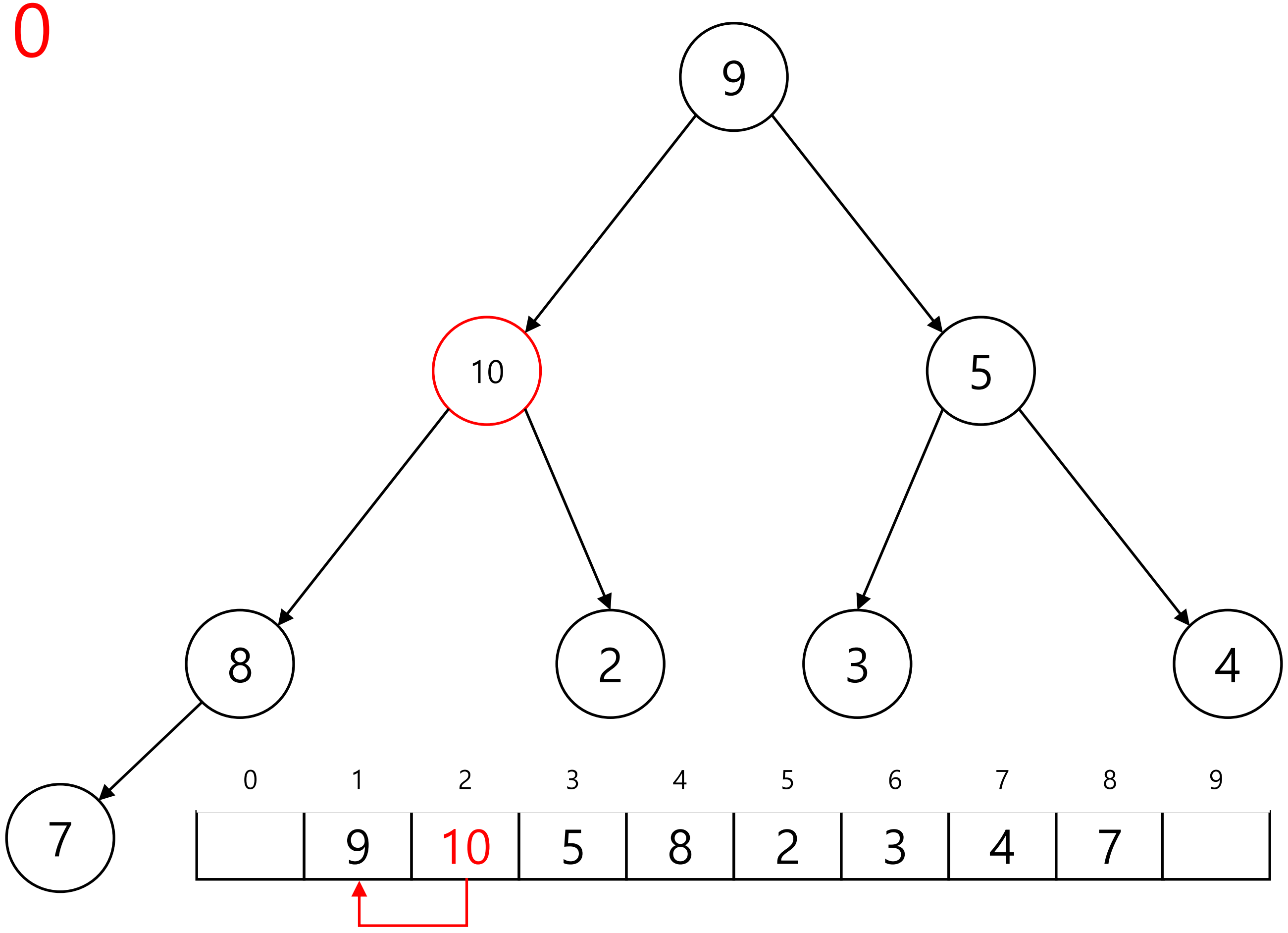
원소 삽입

- INSERT 10



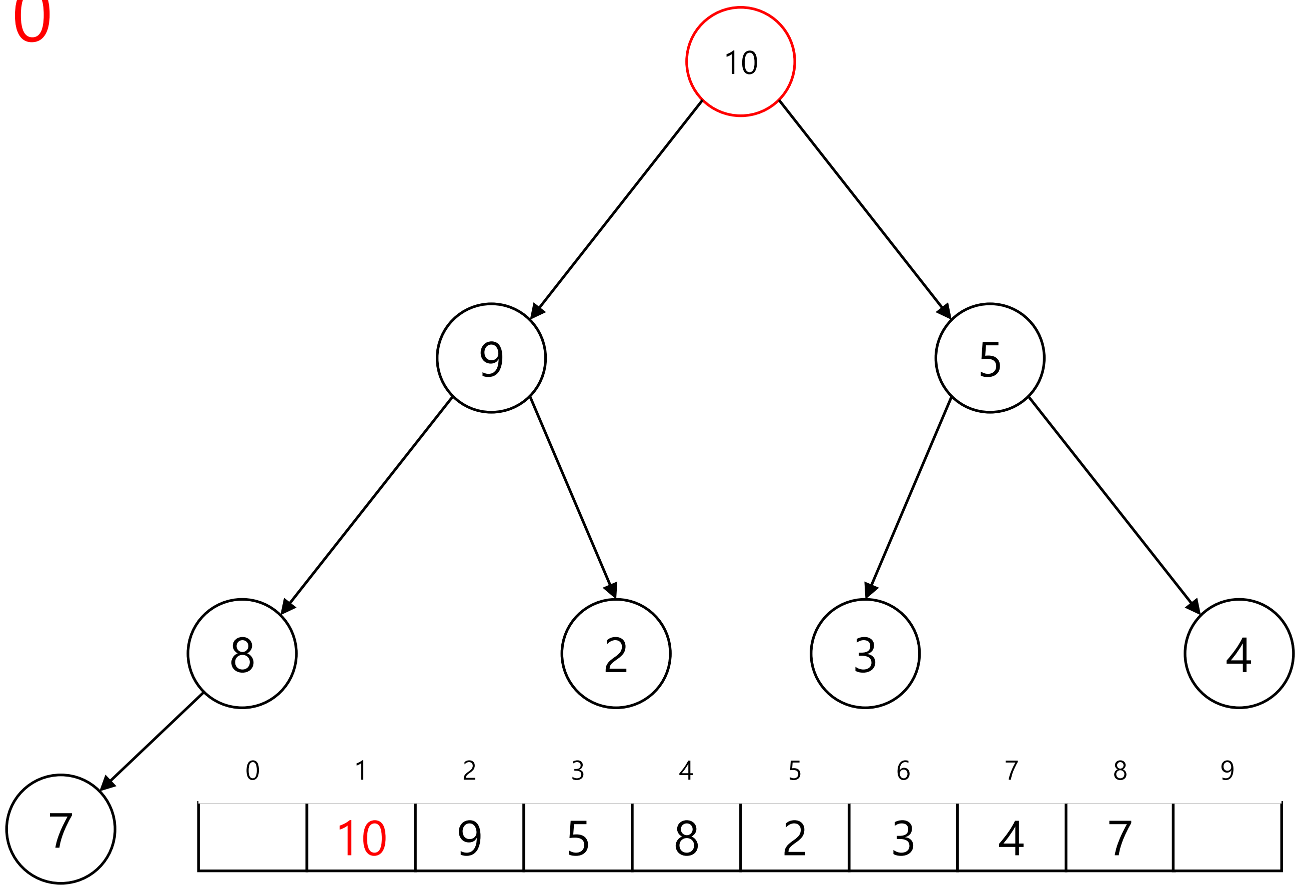
원소 삽입

- INSERT 10



원소 삽입

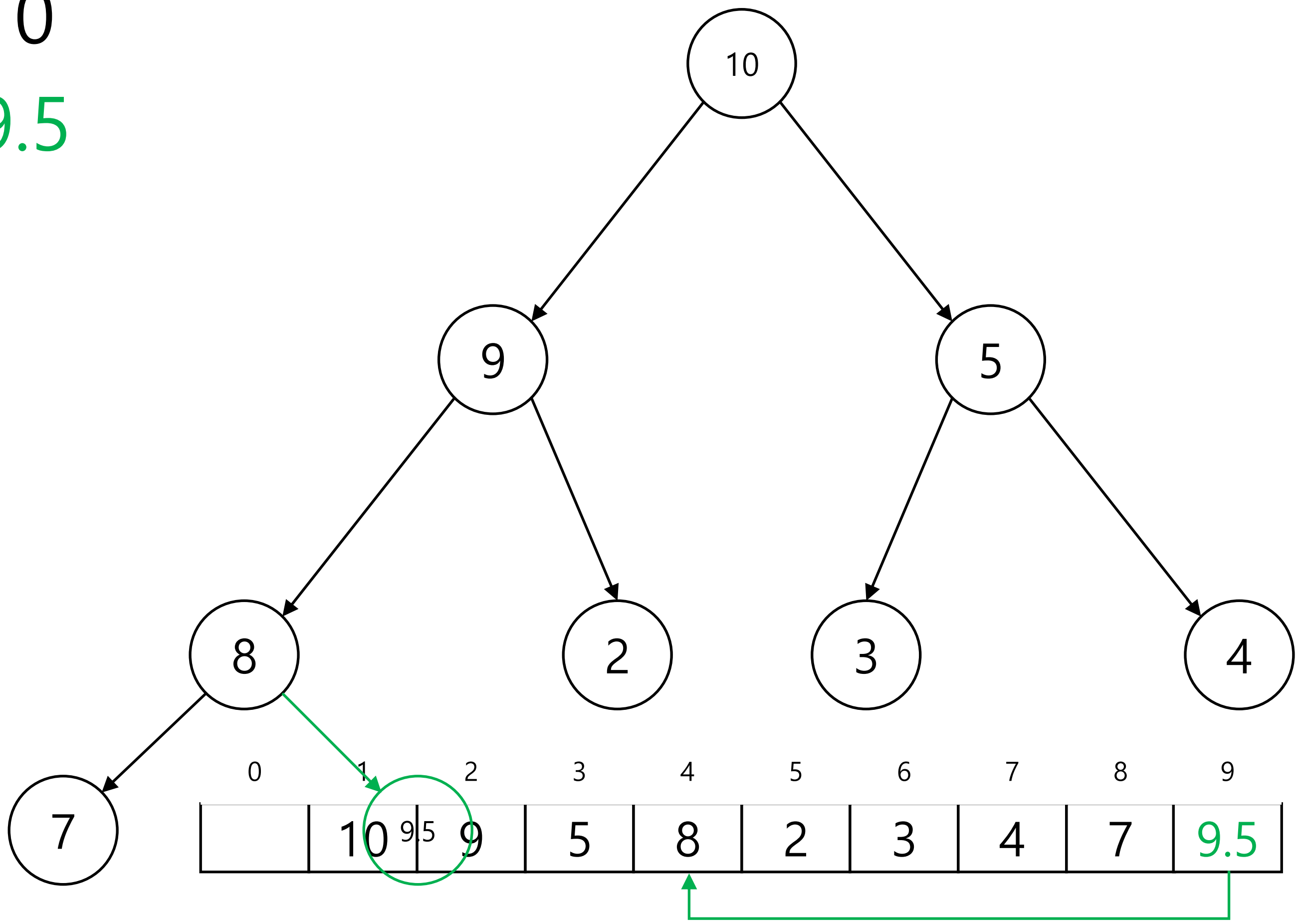
- INSERT 10



0	1	2	3	4	5	6	7	8	9
	10	9	5	8	2	3	4	7	

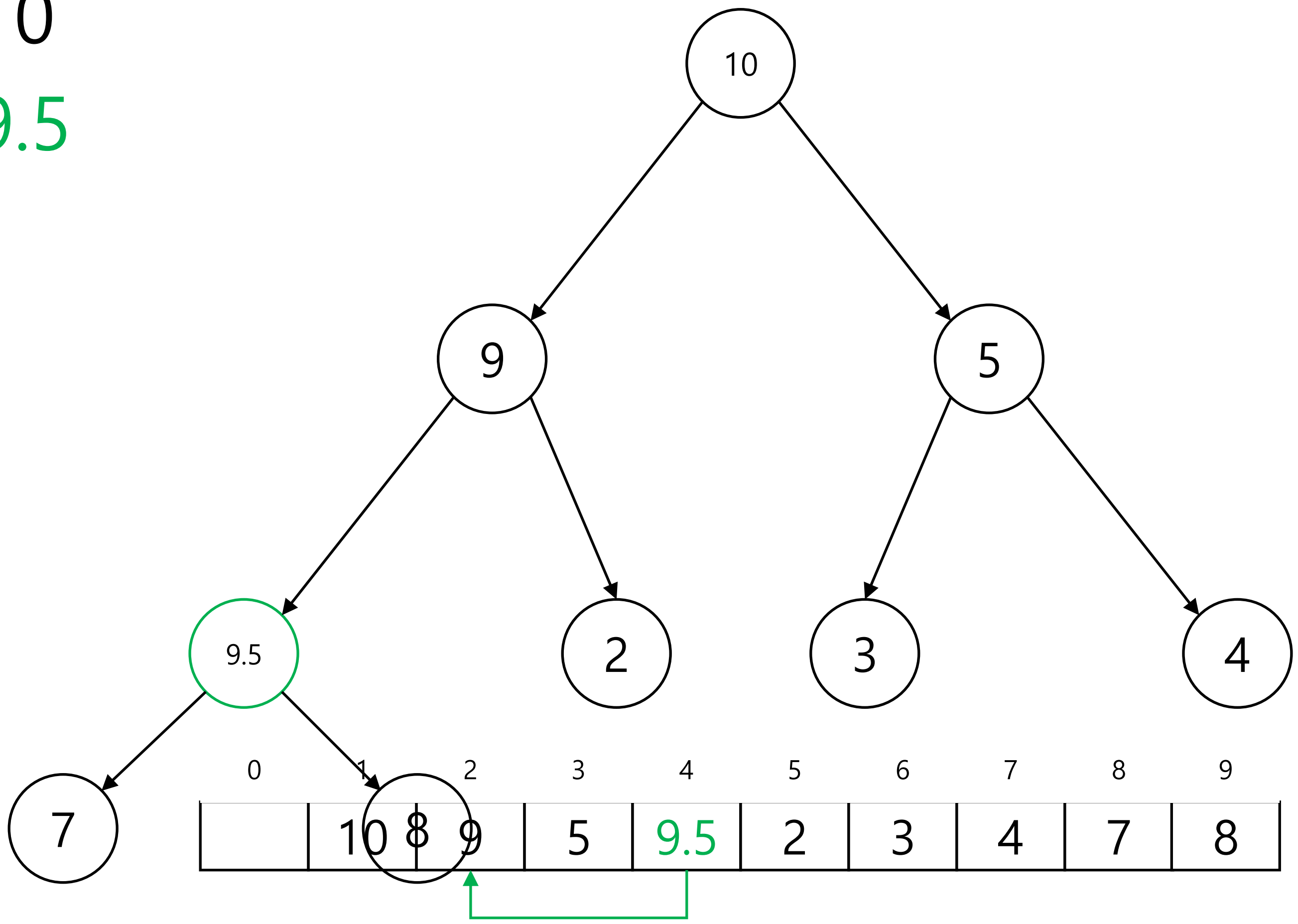
원소 삽입

- INSERT 10
- INSERT 9.5



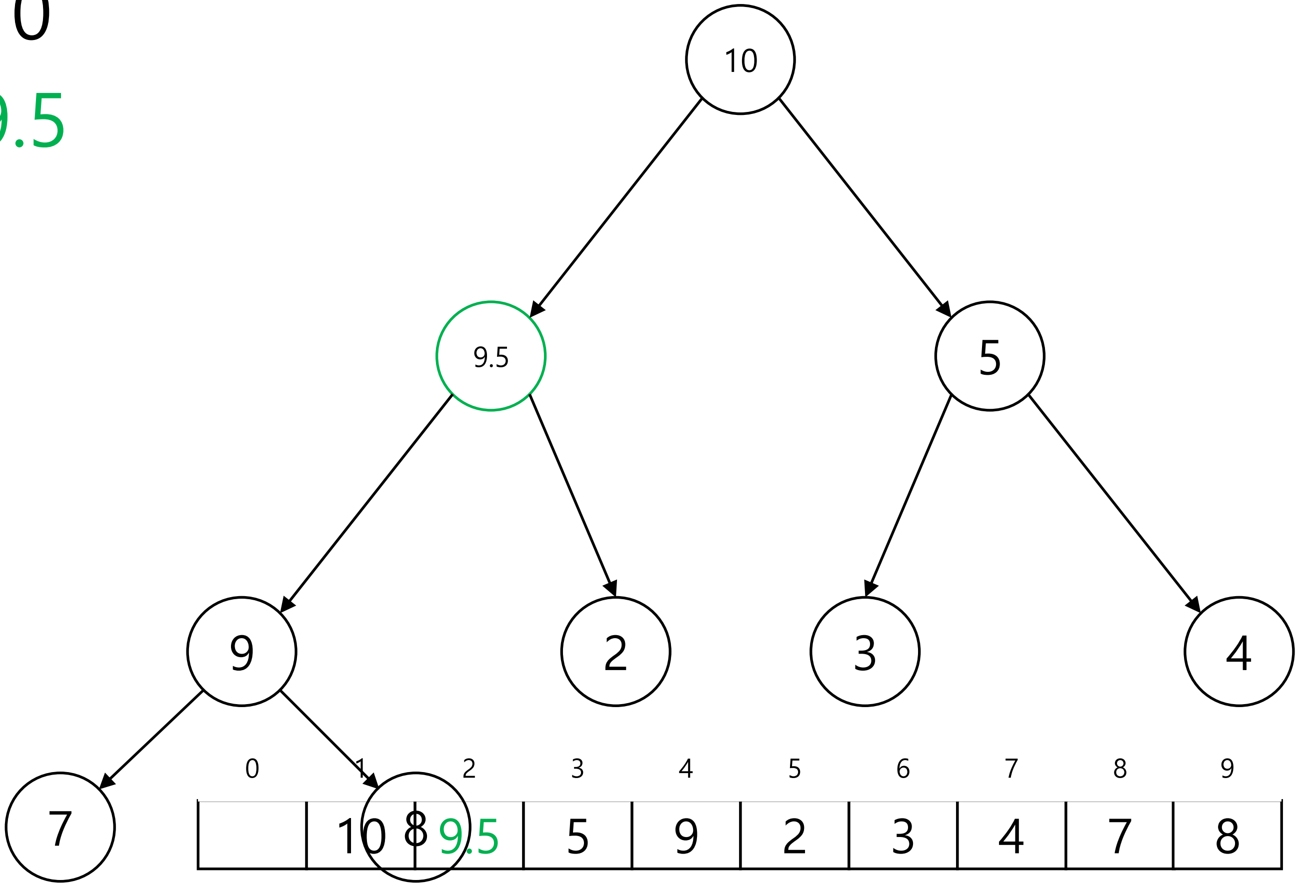
원소 삽입

- INSERT 10
- INSERT 9.5



원소 삽입

- INSERT 10
- INSERT 9.5



시간 복잡도

- 트리의 높이를 h라고 하면 O(h) = O(log N)

가장 큰 원소 탐색

- return heap[1];
- 시간 복잡도 : O(1)

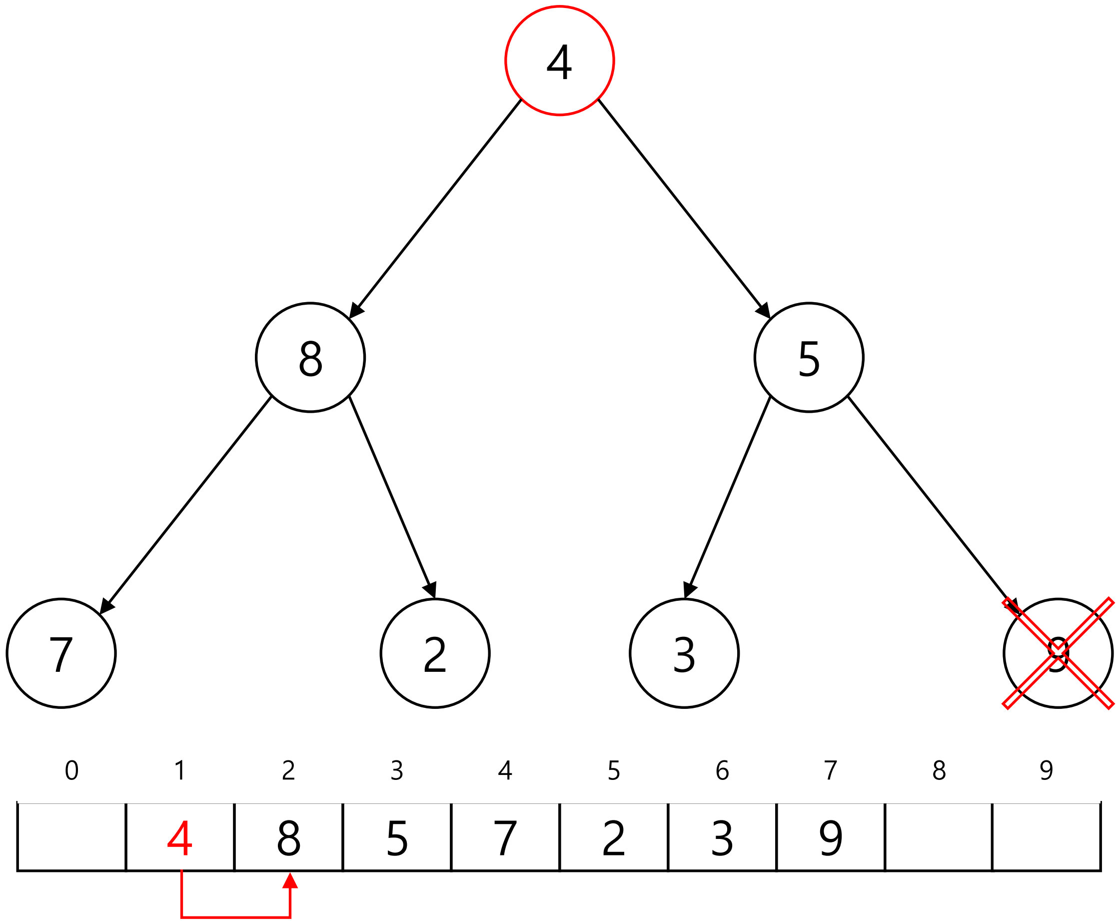
질문?

가장 큰 원소 제거

- 루트(가장 큰 원소)와 마지막 정점의 값을 바꿈
- 마지막 정점 제거
- 현재 루트에 있는 값이 자식보다 작다면 밑으로 내림
  - 두 자식 모두 현재 정점보다 크다면 더 큰 방향으로 내려감
- 반복…

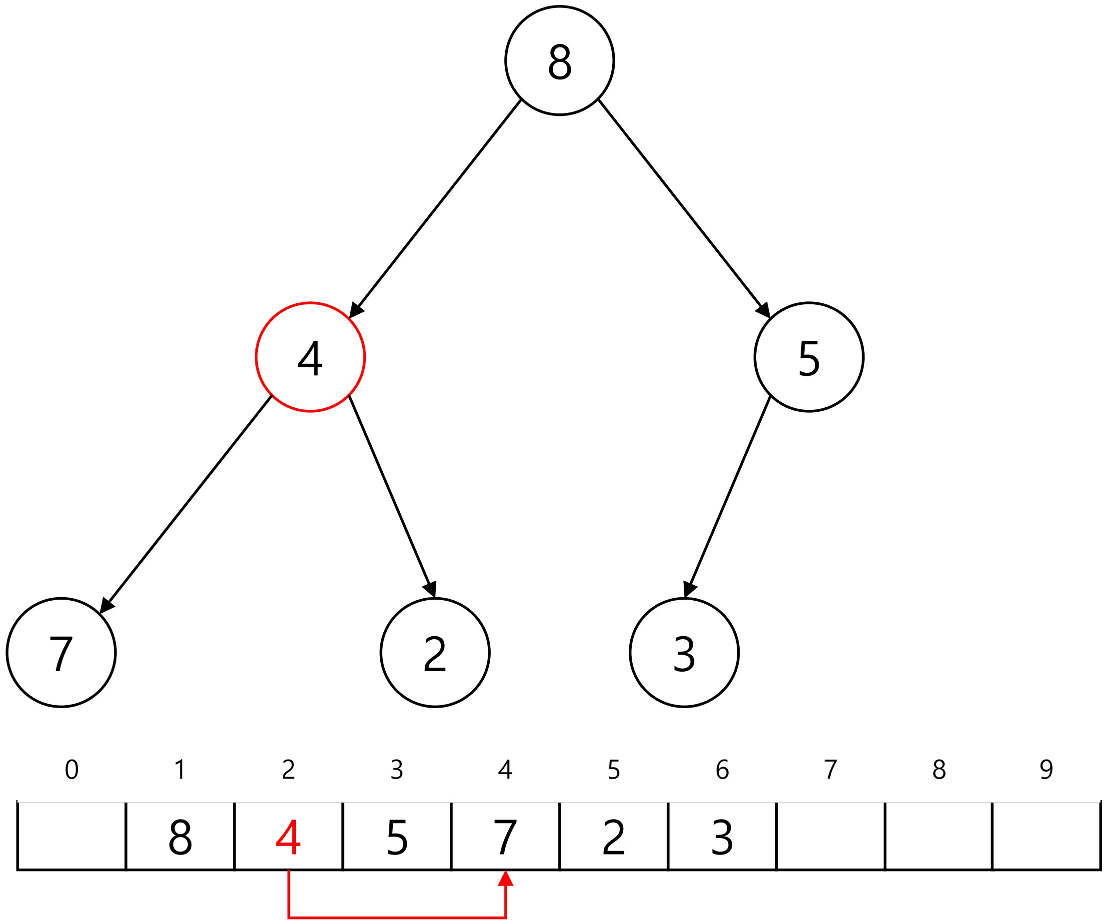
가장 큰 원소 제거

- POP (9)



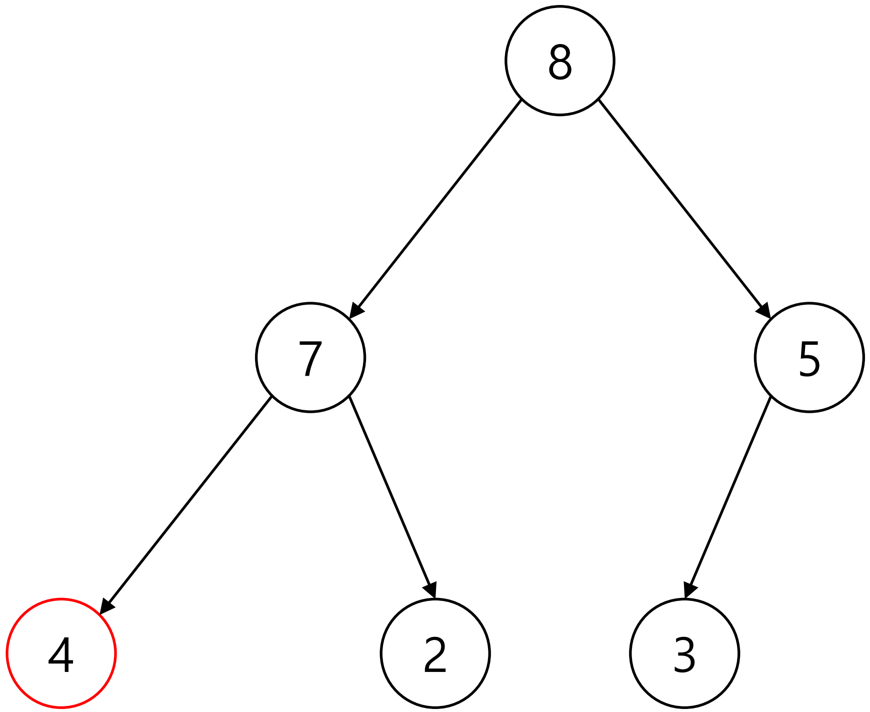
가장 큰 원소 제거

- POP (9)



가장 큰 원소 제거

- POP (9)



0	1	2	3	4	5	6	7	8	9
	8	7	5	4	2	3			

시간 복잡도

- 트리의 높이를 h라고 하면 O(h) = O(log N)

질문?

구현

```
int Heap[101010], sz = 0;

void push(int v){
    Heap[++sz] = p;
    for(int i=sz; i>1; i/=2){
        if(Heap[i] > Heap[i/2]) swap(Heap[i/2], Heap[i]);
        else break;
    }
}

void pop(){
    Heap[1] = Heap[sz--];
    for(int i=1; i*2<=sz; ){
        int ch = i * 2;
        if(ch+1 <= sz && Heap[ch+1] > Heap[ch]) ch += 1;
        if(Heap[ch] > Heap[i]) swap(Heap[ch], Heap[i]), i = ch;
        else break;
    }
}
```

std::prirority_queue

- C++에는 Heap이 이미 구현되어 있고, STL로 제공됨

질문?

유니온 파인드

서로소 집합

- 서로소 집합 : 교집합이 공집합
  - 두 집합 A, B가 A = B이거나 $A \cap B = \emptyset$이면 서로소 집합이다.
  - 서로소인 두 정수 A, B의 소인수분해를 생각해보면 이해하기 쉽다.

Union-Find

- Union-Find : 서로소 집합을 관리하는 자료구조
  - Init : 모든 원소가 자기 자신만을 원소로 하는 집합에 속하도록 초기화
  - Union(u,v) : u, v가 각각 속한 집합을 한 집합으로 병합
  - Find(v) : v가 포함되어 있는 집합을 반환

- 위 3가지 연산을 빠르게 구현하는 것이 목적이다.

Rooted Tree 표현

- 각각의 집합을 Rooted Tree로 표현한다고 생각해보자.
  - Init은 Forest에서 모든 간선을 제거하는 것과 동일하다.
  - Find(v)는 v가 속한 Tree의 루트 정점을 반환하면 된다.
  - Union(u,v)는 u가 속한 Tree의 루트를 v가 속한 Tree의 루트의 자식으로 넣어주면 된다.

- 시간 복잡도는?

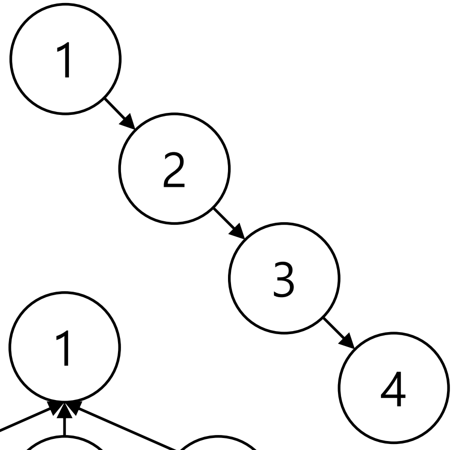
Rooted Tree 표현

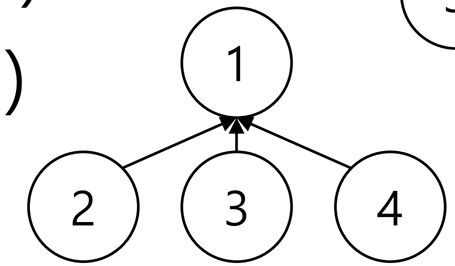
```
int Parent[101010];

void Init(int n){
    for(int i=1; i<=n; i++) return Parent[i] = i;
}

int Find(int v){
    if(v == Parent[v]) return v;
    return Find(Parent[v]);
}

void Union(int u, int v){
    int u_root = Find(u), v_root = Find(v);
    if(u_root != v_root) Parent[u_root] = v_root;
}
```

시간 복잡도

- Init : O(N)
- Find : 트리의 높이를 h라고 하면 O(h)
- Union : Find와 동일

- 최악의 경우 O(N)
- 최선의 경우 O(1)





질문?

최적화

- 트리의 높이를 줄여야 함
  - Union by Rank
  - Union by Size
  - Path Compression
  - …

  - 3개 중 하나만 써도 amortized O(log N)이 보장 된다.
    - 위에 2개는 O(log N)
    - 마지막은 amortized O(log N)
  - Union by Rank과 Path Compression을 사용하면 amortized O(log* N)

Union by Rank

- Rank[i] : i를 루트로 하는 트리의 높이의 상한
- 높이가 낮은 트리를 높은 트리 아래로 넣어주는 방법
  - 만약 두 트리의 높이가 같다면 Union 후 Rank가 1 증가한다.

```
int Parent[101010], Rank[101010];

void Init(int n){
    for(int i=1; i<=n; i++) return Parent[i] = i, Rank[i] = 1;
}

int Find(int v){
    if(v == Parent[v]) return v;
    return Find(Parent[v]);
}

void Union(int u, int v){
    int u_root = Find(u), v_root = Find(v);
    if(u_root == v_root) return;
    if(Rank[u_root] > Rank[v_root]) swap(u_root, v_root);
    Parent[u_root] = v_root;
    if(Rank[u_root] == Rank[v_root]) Rank[v_root]++;
}
```

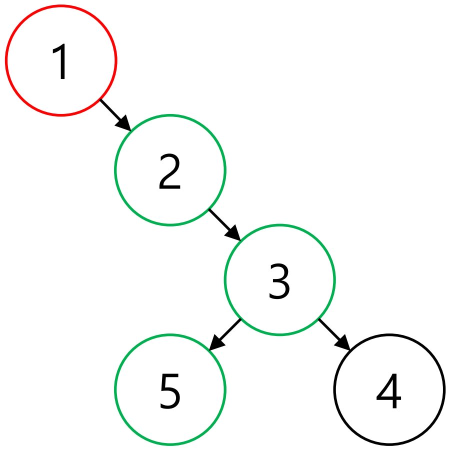
Union by Size

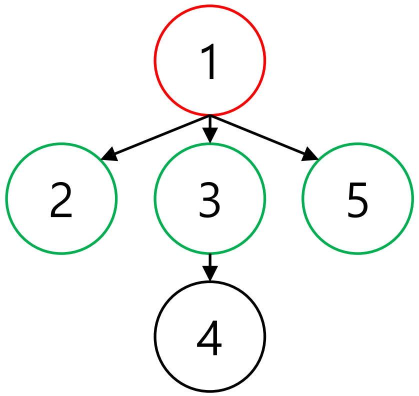
- Size[i] : i를 루트로 하는 트리의 정점 개수
- 정점이 적은 트리를 많은 트리 아래로 넣어주는 방법

```
int Parent[101010], Size[101010];

void Init(int n){
    for(int i=1; i<=n; i++) return Parent[i] = i, Size[i] = 1;
}

int Find(int v){
    if(v == Parent[v]) return v;
    return Find(Parent[v]);
}

void Union(int u, int v){
    int u_root = Find(u), v_root = Find(v);
    if(u_root == v_root) return;
    if(Size[u_root] > Size[v_root]) swap(u_root, v_root);
    Parent[u_root] = v_root;
    Size[v_root] += Size[u_root];
}
```

Path Compression

- 말 그대로 경로 압축
  - Find를 통해 루트를 찾았다면
  - 해당 경로 상에 있는 정점을 모두 루트 바로 밑(자식)으로 붙여줌






```
int Parent[101010];

void Init(int n){
    for(int i=1; i<=n; i++) return Parent[i] = i;
}

int Find(int v){
    if(v == Parent[v]) return v;
    return Parent[v] = Find(Parent[v]);
}

void Union(int u, int v){
    int u_root = Find(u), v_root = Find(v);
    if(u_root != v_root) Parent[u_root] = v_root;
}
```

질문?

Small to Large

간단한 문제

- 각 원소는 1부터 10만까지의 색깔을 갖고 있음
- 처음에는 모든 원소들이 분리된 상태로 시작
- 아래 2가지 쿼리 처리
  - Union(u, v) : u가 속한 집합과 v가 속한 집합을 합침
  - Find(v) : v가 속한 집합에 있는 원소들의 서로 다른 색깔의 종류 반환

풀이

- Union-Find를 사용한다.
  - 각 정점마다 집합의 색깔을 std::set으로 관리
  - Union(u, v)에서 두 set를 합친다.

- 최악의 경우 O(N^2 log N), log N은 set의 시간 복잡도
  - {1}에 {2} 넣고
  - {1,2}에 {3} 넣고
  - {1,2,3}에 {4} 넣고
  - {1,2,3,4}에 {5} 넣고
  - …

Small to Large

- Idea: 작은 집합에 있는 원소를 큰 집합으로 옮기면 커팅 가능
  - 얼마나 줄일 수 있을까?
  - O(N^2)이 O(N log N)이 된다!
  - Why?

  - 처음에는 모든 집합의 크기가 1
  - 작은 집합의 값을 큰 집합으로 이동하기 때문에
    - 원소 v가 다른 집합으로 이동할 때마다
    - v가 속한 집합의 크기가 2배 이상 증가
  - 집합의 최대 크기는 N이므로 각 원소는 최대 O(log N)번 이동 가능
  - 원소의 이동 횟수는 O(N log N)

정리

- 항상 작은 집합에 있는 원소를 큰 집합으로 옮기면 O(N log N)
    - 이 방법을 응용해서 Union by Size가 O(log N)을 보장함을 증명해보자.
    - ↑숙제임

질문?

Sparse Table

재미있는 함수 문제

- 정의역과 치역이 모두 자연수인 함수 $y = f(x)$가 주어진다.
- 다음과 같은 쿼리를 효율적으로 처리하자.
  - $f^k(x)$: x에 f를 k번 적용한 값을 출력한다.
    - ex) $f^1(x) = f(x)$, $f^2(x) = f(f(x))$, $f^3(x) = f(f(f(x)))$
  - $k \le$ 10억

- $O(k)$는 어림도 없다.

이진법

- k = 12라고 하자. 이진법으로 나타내면 1100(2) = 8+4이다.
- x에 f^8을 적용하고, f^4를 다시 적용하면 된다.

- 만약 f^(2^i)꼴을 모두 알고 있다면?
  - 각 쿼리를 O(log k)에 처리할 수 있다!

Sparse Table

- P[i][j] = f^(2^i)(j)
  - j에서 2^i번 만큼 이동한 결과
- P[0][j] = f(j)
  - 2^0 = 1
- P[i][j] = P[i-1][P[i-1][j]]
  - j에서 2^i번 만큼 이동한 결과
  - j에서 2^(i-1)번 이동하고, 다시 2^(i-1)번 이동한 결과

- k, j의 최댓값이 K, N이면 O(N log K)에 전처리 가능

쿼리 처리

- 비트 연산을 잘 쓰면 된다.

```
int Query(int k, int v){
    for(int i=MX_BIT-1; i>=0; i--){
        if((k >> i) & 1) v = P[i][v];
        // if(k & (1 << i)) v = P[i][v];
    }
    return v;
}
```

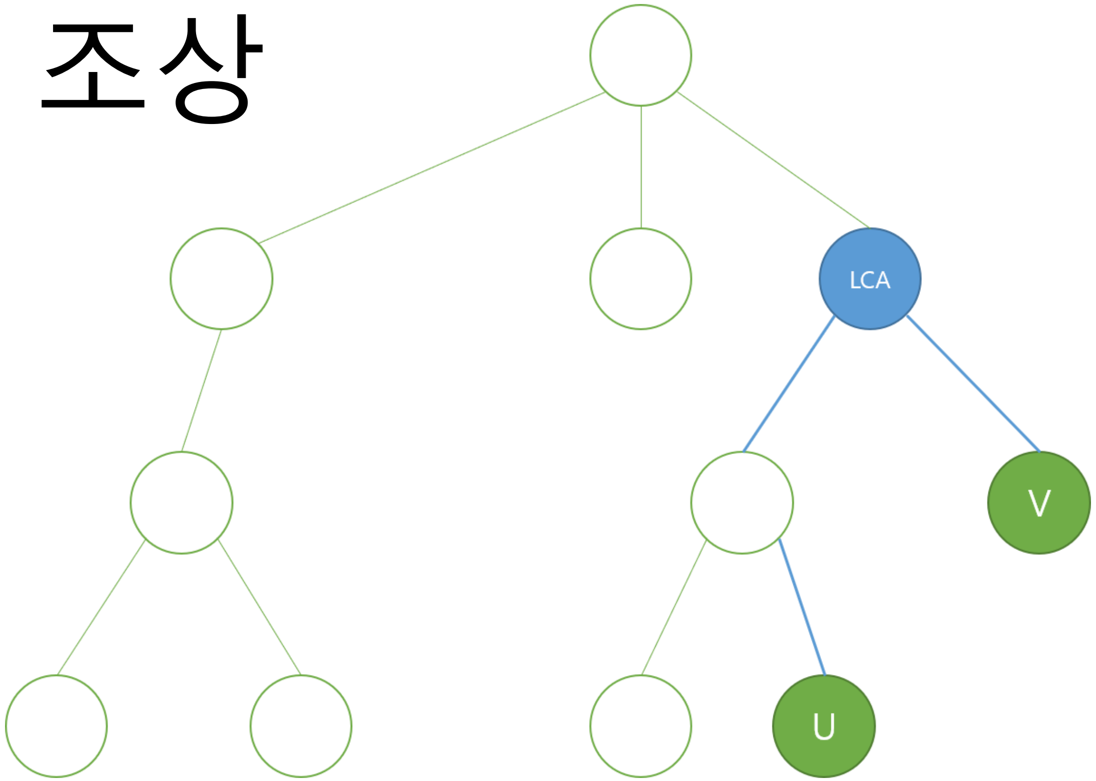
질문?

LCA

최소 공통 조상

- LCA : Lowest Common Ancestor
  - Lowest : 가장 낮은
  - Common : 공통
  - Ancestor : 조상

- LCA(u, v)
  - 트리의 정점 u, v의 공통되는 조상 중 가장 아래에 있는 정점



간단한 방법

- u와 v의 깊이가 다른 경우, 깊이를 동일하게 맞춘다.
- u와 v가 동시에 한 칸 씩 올라가면서 처음으로 u = v가 되는 지점이 LCA

- 시간 복잡도 : O(h) + O(h) = O(h) ⇒ O(N)

효율적인 방법

- 앞에서 소개한 두 가지 작업을 각각 O(log N)에 처리함

- 깊이를 맞추는 작업
  - 깊이의 차이를 k이고, v가 더 깊이 있다고 하자.
  - v의 k번째 조상을 찾으면 된다.
    - P[i][v]를 v의 2^i번째 조상이라고 정의하면
    - Sparse Table이므로 O(log N)

효율적인 방법

- 앞에서 소개한 두 가지 작업을 각각 O(log N)에 처리함

- 동시에 한 칸 씩 올라가는 작업
  - Parametric Search를 한다.
  - u ≠ v인 가장 높은 지점을 찾으면, 그 지점의 부모가 LCA가 된다.
    - 첫 번째 수업에서 잠깐 소개한 방법과 Sparse Table을 사용하면 O(log N)

정수 범위라면 이것도 가능하다!

- 첫 번째 단계에서 m = 2^(K-1)
- 두 번째 단계에서 m = 2^(K-2)
- i 번째 단계에서 m = 2^(K-i)

```
constexpr int K = 18;
int Maximize(){
    // int l = 0, r = (1 << K) - 1;
    int ans = 0;
    for(int bit=K-1; bit>=0; bit--){
        if(Decision(ans | 1 << bit)) ans |= 1 << bit;
    }
    return ans;
}
```

질문?

코드

```cpp
#include <bits/stdc++.h>
using namespace std;

int N, Q, D[101010], P[22][101010];
vector<int> G[101010];

void DFS(int v, int b=-1){
    for(auto i : G[v]){
        if(i == b) continue;
        D[i] = D[v] + 1; P[0][i] = v;
        DFS(i, v);
    }
}

int LCA(int u, int v){
    if(D[u] < D[v]) swap(u, v);
    int diff = D[u] - D[v];
    for(int i=0; diff; i++, diff>>=1) if(diff & 1) u = P[i][u];
    if(u == v) return u;
    for(int i=21; i>=0; i--) if(P[i][u] != P[i][v]) u = P[i][u], v = P[i][v];
    return P[0][u];
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<N; i++){
        int s, e; cin >> s >> e;
        G[s].push_back(e); G[e].push_back(s);
    }
    DFS(1);
    for(int i=1; i<22; i++) for(int j=1; j<=N; j++) P[i][j] = P[i-1][P[i-1][j]];
    cin >> Q;
    while(Q--){
        int u, v; cin >> u >> v;
        cout << LCA(u, v) << "\n";
    }
}
```

질문?